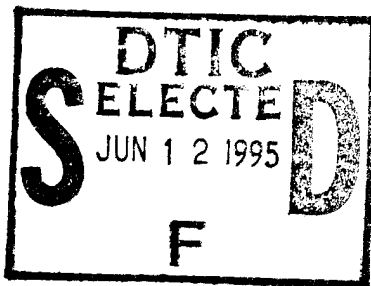# NATIONAL AIR INTELLIGENCE CENTER

C FUNCTION RECOGNITION TECHNIQUE AND ITS IMPLEMENTATION
IN 8086 C DECOMPILING SYSTEMS

by

Chen Fuan, Liu Zongtian

19950608 004

**Approved for public release:
distribution unlimited**

HUMAN TRANSLATION

NAIC-ID(RS)T-0023-95   26 May 1995

MICROFICHE NR: 95C000334

C FUNCTION RECOGNITION TECHNIQUE AND ITS IMPLEMENTATION
IN 8086 C DECOMPILING SYSTEMS

By: Chen Fuan, Liu Zongtian

English pages: 20

Source: Xiaoxing Weixing Jisaunji, Vol. 12, Nr. 11,
         1991; pp. 33-40; 47

Country of origin: China
Translated by: SCITRAN
                F33657-84-D-0165
Requester: NAIC/TATA/Keith D. Anthony
Approved for public release: distribution unlimited.

GRAPHICS DISCLAIMER

All figures, graphics, tables, equations, etc. merged into this
translation were extracted from the best quality copy available.

Accesion For		
NTIS CRA&I		☑
DTIC TAB		☐
Unannounced		☐
Justification		
By		
Distribution /		
Availability Codes		
Dist	Avail and / or Special	
A-1		

1

C FUNCTION RECOGNITION TECHNIQUE AND ITS IMPLEMENTATION IN 8086 C DECOMPILING SYSTEMS

Chen Fuan   Liu Zongtian

ABSTRACT

In 8086 C decompilation systems, we take pattern recognition principles and apply them to C function recognition. This article, first of all, analyzes C function component characteristics in object code. Following that, it introduces library function pattern recognition techniques in 8086 C decompilation systems, that is, C function recognition characteristic extraction, recognition pattern matching methods, and the establishment of library function characteristic code recognition tables. Finally, it introduces implementation methods associated with the library function recognition techniques in question in 8086 C decompilation systems.

I.   INTRODUCTION

Decompilation system functions are to take machine language object code programs and transform them into functionally equivalent high level language programs, causing programs to be convenient to read, understand, modify, and protect. Decompilation acts as a type of tool for software analysis and understanding. It is a key component part associated with software reverse work processes. It possesses important practical value.

We developed 68000 C decompilation systems on Dual-68000 micro machines [1]. The systems in question are capable of taking C compilation formed 68000 object code programs (containing symbol information such as overall variable names as well as library function names, and so on) and transforming them into C language programs. However, the majority of object code programs in actual use are ones which do not contain any symbol information. In order to increase the practicality of decompilation techniques, on PC machines, we developed C language decompilation systems aimed at 8086 object code programs not /34 carrying any symbol information and formed by C compilation (this

1

item is a subtask of national item 7-5, serial no. 68-4-3/01).
As a result, there exist symbol restoration problems associated
with variable names and library function names. On the
foundation of component analysis carried out on object code
programs formed by C compilation on PC machines, the symbol
restoration plan we opted for the use of is: during
decompilation processes associated with object code programs,
apply pattern recognition principles to carry out C function
recognition, thereby restoring library function symbol names
utilized in programs, form compilation language programs having
library function names, and, as far as restored variable names
and data types are concerned, put them back into outcome symbol
tables to construct phases of execution.

II. C FUNCTION COMPONENT ANALYSIS IN OBJECT CODE PROGRAMS
     In order to recognize library functions in object code
programs, first of all, one should understand the structural
forms associated with object code program code composition,
understand the forms of utilization associated with library
functions in object code programs, and analyze code composition
characteristics associated with library functions.
     1. Structural Forms Associated with Object Code Program
Code Composition
     On IBM PC/AT micro machines, we dissected and analyzed
object code programs formed by compilation with small memory
patterns from Microsoft C(Ver.5.0) [3]. The code composition
structural forms are:
     (1) From offset address 10H to offset addresses associated
with the beginning of program execution, one reduced storage unit
is the machine code storage area associated with main function as
well as user definition functions in C original programs--we
designate it as program code area (see as shown in Fig.1).

```
__main:
{
  48AF:0010  55           PUSH   BP
  48AF:0011  8BEC         MOV    BP,SP
  48AF:0013  33C0         XOR    AX,AX
  48AF:0015  E88602       CALL   __chkstk (029E)
  48AF:0018  E81C00       CALL   __function (0037)
  48AF:001B  FF364200     PUSH   Word Ptr [0042]
  48AF:001F  B87402       MOV    AX,0274
  48AF:0022  50           PUSH   AX
  48AF:0023  E8C005       CALL   __printf (05E6)
  48AF:0026  83C404       ADD    SP,+04
  48AF:0029  B80500       MOV    AX,0005
  48AF:002C  50           PUSH   AX
  48AF:002D  E87011       CALL   __close (11A0)
  48AF:0030  83C402       ADD    SP,+02
  48AF:0033  8BE5         MOV    SP,BP
  48AF:0035  5D           POP    BP
  48AF:0036  C3           RET
}

__function:
{
  48AF:0037  55           PUSH   BP
  48AF:0038  8BEC         MOV    BP,SP
  48AF:003A  33C0         XOR    AX,AX
  48AF:003C  E85F02       CALL   __chkstk (029E)
  48AF:003F  B87802       MOV    AX,0278
  48AF:0042  50           PUSH   AX
  48AF:0043  FF364200     PUSH   Word Ptr [0042]
  48AF:0047  E8A815       CALL   __strcmp (15F2)
  48AF:004A  83C404       ADD    SP,+04
  48AF:004D  A38007       MOV    Word Ptr [0780],AX
  48AF:0050  8BE5         MOV    SP,BP
  48AF:0052  5D           POP    BP
  48AF:0053  C3           RET
}
```

```c
#include <stdio.h>
#include <io.h>

int  x;
char * str="string";

main( )
{
    function( );
    printf("%s",str);
    close(5);
}

function( )
{
    x = strcmp(str,"string");
}
```

(b)   Original Program
      Example

(a) Program Code Area Corresponding
     to Original Program

Fig.1 Program Code Area Example

   (2) Memory elements from program execution start addresses
to the approximately 150 character segments which follow are
machine code storage areas associated with common operations
formed in C compilation--we designate them as main control code
areas. The code zones in question, in various object code
programs, all are stored and function the same.   /35

   (3) Following main control code areas--to the conclusion of
program code sections--are subprogram machine code storage areas
associated with library functions or systems utilized during
program execution. We designate all of these as library function
code areas.

3

Structural forms associated with object code program code composition are as shown in Fig.2.

On the basis of the structural characteristics associated with object code program code composition--comparing it to C language program composition--in 8086 C decompilation systems, we only need to take program code areas which are decompiled into compilation language programs which correspond to main functions and user defined functions, and, in conjunction with this, go through main control code areas and find main function start offset addresses as well as utilizing library function code areas in order to recognize library functions utilized in programs.
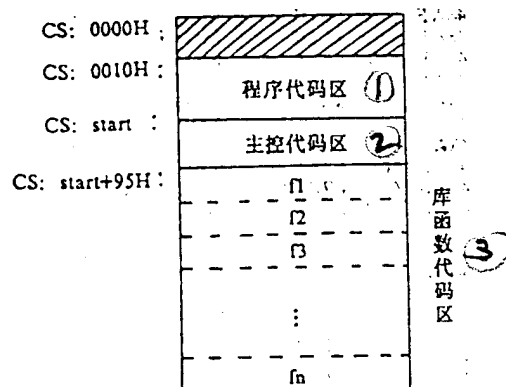

Fig.2 Structural Forms Associated with Object Code Program Code Composition

Key: (1) Program Code Area (2) Main Control Code Area
(3) Library Function Code Area

2. Composition Forms Associated with C Library Functions in Object Code Programs

Through analysis of object code programs formed by C compilation, it is discovered that C library function utilization forms in object code programs as well as code composition characteristics are:

1) C library functions (excluding macro definition functions) all use subprogram forms to appear in object code programs. Library functions used in user programs all have subprogram code sections associated with corresponding functions in object code programs after going through compilation. In the same program, the same library function is used in multiple locations. However, corresponding library function code sections in object code programs, only appear once. The majority of subprograms corresponding to library functions are associated with insertion into sets and transfers.

2) C library function code is not influenced by compilation optimization. Moreover, library function code in the same program is not influenced by input parameters using different forms. However, in different programs, the same library function is used. Due to changes associated with variable parameter distribution addresses as well as transferred subprogram assembly addresses during compilation and linking assembly, it will lead to command codes to these addresses associated with certain uses in library function code to also produce variations. However, this is only command operation numbers producing changes. By contrast, operation code will not change. Therefore, command operation code sequences in library function code will not, under any circumstances, produce variations (see Fig.3 example).

(3) Subprograms corresponding to C library functions used in programs have transfer input addresses which, in all cases,

5

appear in program code areas. They appear in the form of CALL Subroutine___address commands. In this way, we are capable, with regard to program code area decompilation processes, of searching out various individual library function input addresses utilized for library function recognition uses (see Fig.1 example).

Starting out from the angle of library function recognition, we use the two standards of storage forms associated with library function code (main subprogram portion) in object programs and whether or not operation numbers in library function code contain variations. It is possible to take composition forms associated with C library function code and divide them into the three basic types below:

Continuous Not Deformed--This type of library function code is continuously stored in object code programs. Code does not contain such commands as use variable parameter addresses, subprogram transfers, or absolute transfers. As a result, no matter in what type of situation utilization is, library function code, in all cases, will not produce variations--for example, strcmp function code as shown in Fig.3(a).

Continuous Deformable--This type of library function code in object code programs is continuously stored. However, within code, there are contained certain commands such as utilization variable parameter addresses, subprogram transfers, and so on. As a result, following along with changes in variable distribution addresses as well as assembly addresses for transferred subprograms, operation number code corresponding to these commands will also produce variations--for example, code associated with printf function main subprogram sections as shown in 3(b).

Sectioned Deformable--This type of library function code is divided into certain sections and stored. The various sections

6

are put through transfer type functions for the sake of
continuous execution. Basically, the code associated with the
various sections belongs to a deformable type--for example, the
code associated with close function main subprogram sections as
shown in Fig.3(c).   /36

```
__sgrcmp:
48AF:15F2  55        PUSH   BP
48AF:15F3  8BEC      MOV    BP,SP
48AF:15F5  8BD7      MOV    DX,DI
48AF:15F7  8BDE      MOV    BX,SI
48AF:15F9  8CD8      MOV    AX,DS
48AF:15FB  8EC0      MOV    ES,AX
48AF:15FD  8B7604    MOV    SI,Word Ptr [BP+04]
48AF:1600  8B7E06    MOV    DI,Word Ptr [BP+06]
48AF:1603  33C0      XOR    AX,AX
48AF:1605  B9FFFF    MOV    CX,FFFF
48AF:1608  F2AE      REPNE  SCASB
48AF:160A  F7D1      NOT    CX
48AF:160C  2BF9      SUB    DI,CX
48AF:160E  F3A6      REPZ   CMPSB
48AF:1610  7405      JZ     __strcmp+25 (1617)
48AF:1612  1BC0      SBB    AX,AX
48AF:1614  1DFFFF    SBB    AX,FFFF
48AF:1617  8BF3      MOV    SI,BX
48AF:1619  8BFA      MOV    DI,DX
48AF:161B  5D        POP    BP
48AF:161C  C3        RET
```

(a)__sgrcmp code

```
__printf:
48AF:05E6  55        PUSH   BP
48AF:05E7  8BEC      MOV    BP,SP
48AF:05E9  83EC08    SUB    SP,+08
48AF:05EC  57        PUSH   DI
48AF:05ED  56        PUSH   SI
48AF:05EE  BE1001    MOV    SI,0110
48AF:05F1  8D4606    LEA    AX,Word Ptr [BP+06]
48AF:05F4  8946FC    MOV    Word Ptr [BP-04],AX
48AF:05F7  56        PUSH   SI
48AF:05F8  E8BD01    CALL   __stbuf   (07B8)
48AF:05FB  83C402    ADD    SP,+02
48AF:05FE  8BF8      MOV    DI,AX
48AF:0600  8D4606    LEA    AX,Word Ptr [BP+06]
48AF:0603  50        PUSH   AX
48AF:0604  FF7604    PUSH   Word Ptr [BP+04]
48AF:0607  56        PUSH   SI
48AF:0608  E85103    CALL   __output   (095C)
48AF:060B  83C406    ADD    SP,+06
48AF:060E  8946F8    MOV    Word Ptr [BP-08],AX
48AF:0611  56        PUSH   SI
48AF:0612  57        PUSH   DI
48AF:0613  E85002    CALL   __ftbuf   (0866)
48AF:0616  83C404    ADD    SP,+04
48AF:0619  8B46F8    MOV    AX,Word Ptr [BP-08]
48AF:061C  5E        POP    SI
48AF:061D  5F        POP    DI
48AF:061E  8BE5      MOV    SP,BP
48AF:0620  5D        POP    BP
48AF:0621  C3        RET
```

(b)   __printf main subprogram section code

```
__close:
48AF:11A0  55          PUSH   BP
48AF:11A1  8BEC        MOV    BP,SP
48AF:11A3  8B5E04      MOV    BX,Word Ptr [BP+04]
48AF:11A6  3B1EC200    CMP    BX,Word Ptr [00C2]
48AF:11AA  7206        JB     __close+12   (11B2)
48AF:11AC  B80009      MOV    AX,0900
48AF:11AF  F9          STC
48AF:11B0  EB0B        JMP    __close+1d   (11BD)
48AF:11B2  B43E        MOV    AH,3E
48AF:11B4  CD21        INT    21
48AF:11B6  7205        JB     __close+1d   (11BD)
48AF:11B8  C687C40000  MOV    Byte Ptr [BX+00C4],00
48AF:11BD  E9A0F3      JMP    __dosret0   (0560)
```

```
__dosret0:
48AF:0560  7213        JB     __dosretax+2   (0575)
48AF:0562  33C0        XOR    AX,AX
48AF:0564  8BE5        MOV    SP,BP
48AF:0566  5D          POP    BP
48AF:0567  C3          RET
```

```
__dosretax+2:
48AF:0575  E80E00      CALL   __maperror+6   (0586)
48AF:0578  B8FFFF      MOV    AX,FFFF
48AF:057B  99          CWD
48AF:057C  8BE5        MOV    SP,BP
48AF:057E  5D          POP    BP
48AF:057F  C3          RET
```

(c)__close main
subprogram section
code

Fig.3 C Library Function Code Composition Form Examples

Key: (1) Note: In the Fig., underlined sections represent
operation numbers which can vary.

7

Taking C library codes and dividing them into these three classes of composition type is advantageous to the recovery of library function recognition characteristics and the establishment of pattern matching forms.

III. C LIBRARY FUNCTION PATTERN RECOGNITION TECHNIQUES

1. Recovery of C Library Function Recognition Characteristics

Through analysis, we take command operation code sequences to act as C library function recognition characteristics. This is because:

(1) No matter what composition type C library function codes are or under what conditions they are utilized, their command operation codes will not vary. Only certain command operation numbers are capable of producing changes;

(2) Different library functions--due to differences in their functions--are composed of different command operation code sequences;

(3) On the basis of 8086 command coding rules, in coding where command operation codes contain operation number types and addressing forms, in this way, operation number codes are eliminated from certain command codes. However, the total characteristics of operation number codes certainly have not yet disappeared.

In the recovery of C library function recognition characteristics, the method we opt for the use of is: under the principles of characteristic sufficiency, library function code primary and supplementary characteristics are both taken, that is, recovering primary characteristic codes from main subprogram bodies and also recovering supplementary characteristic codes

from the transfer function subprograms (if there are any). This
is because: /37

    1) there are a number of main subprogram sections associated
with library functions which certainly do not reflect the
functions realized, moreover, certain subprograms which they
transfer still represent the functions of library functions;

    2) there are a number of code sections which correspond to
library functions which are very long; so long as there is a
situation which avoids the inclusion of recovered characteristics
code associated with different library functions, there is no
need to take the whole code to act as characteristic recovery.
This will increase the speed of library function pattern
recognition.

    In accordance with characteristic recovery methods
associated with library function recognition discussed above, C
library function code tables were established. For example, with
regard to strcmp functions, we selected the complete command code
to act as the recognition characteristic code. Moreover, with
regard to printf functions, we took the main subprogram code and
eliminated four changed operation number codes to act as the main
characteristic recognition code. However, we selected the
transferred subprogram code to act as supplementary characteristic
recognition code. With regard to close functions, by contrast,
we selected the various code sections composing main subprogram
bodies and eliminated changed operation number codes to act as
the recognition characteristic code (see Fig.3). On the basis of
whether library function code recovery is complete or not,
characteristic codes can be divided into complete types--taking
whole code to act as characteristic code, as well as compressed
types--under the principles of characteristic sufficiency, taking
sections of code to make up characteristic code. With regard to
the three library function characteristic codes which are cited

as examples above, they all belong to the complete type. With
regard to code sections which are too long and most supplementary
characteristics, uniformly, they are handled like compression
types.

2.   Pattern Matching Methods Associated with C Library
Function Recognition

On the basis of C library function code composition types
and characteristic code recovery methods, we opt for the use of
static matching and dynamic matching methods, respectively
designing eight types of C library function code recognition
pattern matching methods.   Among them, there are four types
associated with the handling of complete characteristic codes:

Complete Matching Methods--Opting for the use of static
matching methods, library function characteristic codes
indicating length and library function codes awaiting recognition
in programs are taken to carry out equal length matching.   This
type of matching method is appropriate to handling continuous
nondeformable types of library function recognition.   It also
fits recognition of library functions of continuous deformable
types not containing supplementary characteristic codes.   The
reason is that, among characteristic codes selected by this type
of library function, altered control number code is already not
contained.   We opted for the use of code section complete
matching methods in order to carry out recognition of this type
of library function.

Subprogram Matching Methods--On library function
characteristic code and library function number code which waits
to be recognized--after carrying out subprogram transfer code
investigations--through dynamic execution of subprogram transfer
commands in programs in order to precisely specify the subprogram
entry address, one subsequently takes the entry addresses in

question and compresses them into queues. At the same time, one also takes supplementary characteristic matching pattern addresses (installed in advance) and compresses them into queues.

After waiting on main characteristic pattern matching, supplementary characteristic pattern matching is then carried out. This type of matching method is used in supplementary characteristic pattern processing in continuous deformable type library function recognition.

Direct Section Transfer Matching Methods--After carrying out subprogram transfer code inspections in library function characteristic codes and library function codes awaiting recognition, dynamic execution of transfer commands in programs is gone through in order to get start addresses for follow on code sections. At the same time, preinstalled follow on section matching pattern addresses are also picked up. They are respectively taken to function as the next library function code start address awaiting recognition and utilized in pattern matching as well as library function matching pattern addresses. This type of matching method is used in the handling of transfer situations between main body code sections during recognition of sectioned deformable type library functions.

Branching Section Transfer Matching Methods--This type of method is used in the handling of branching code section transfer situations during sectioned deformable library function recognition. It obtains code start addresses for library functions awaiting recognition and library function matching pattern addresses. In method, it is the same as direct section transfer matching forms. What is different is that, in the forms in question, one takes these two addresses and compresses them into queues, waits until after main body code section pattern recognition and then carries out branching section processing.

With regard to the four types of complete form pattern matching methods, their common points are that amounts of offset associated with characteristic code addresses stored in characteristic code tables and amounts of offset between library function code awaiting recognition and start entry addresses are equal. The four types of pattern matching methods associated with the processing of compressed form characteristic codes are: compressed type complete matching forms; compressed type subprogram matching forms; compressed type direct section transfer matching forms; and, compression type branching section transfer matching forms.

These four types of matching forms, in terms of processing methods and processing objects, are the same as the corresponding complete type matching forms. What is different is that the amounts of offset associated with their characteristic code addresses stored in characteristic code tables and the amounts of offset associated with library function code waiting to be recognized and start entry addresses are not equal.

We utilize the pattern matching forms described above and set up pattern matching form tables associated with the recognition of various individual C library function characteristic codes. The majority of library function recognition pattern matching form tables are composed of several types of matching forms. For example, strcmp functions only utilize complete matching forms. However, printf functions utilize complete matching and subprogram matching forms (see as shown in Fig.5). close functions, by contrast, must utilize complete matching, direct section transfer, and branching section transfer matching forms.

12

3. C Library Function Characteristic Code Recognition
Tables

In order to realize design concepts associated with C
library function pattern matching recognition, in 8086 C
decompilation systems, we set up a C library function
characteristic code recognition table. It is nothing else than a
gathering of library function matching patterns which the system
can recognize. Each table element associated with the table in
question represents a library function matching pattern. In
library function recognition, with regard to matching of table
elements, we opted for the use of sequence search forms.
Therefore, C library function characteristic code recognition
table design is taken to be a single link table structure (see as
shown in Fig.4). This is because, at the present time, there is
no way, from library function code, to find prioritized search
information. We are only able, on the basis of C language
program writing experience, to take common library function
matching patterns and store them in the front of tables. Levels
of prioritized search are reflected in the order of arrangement
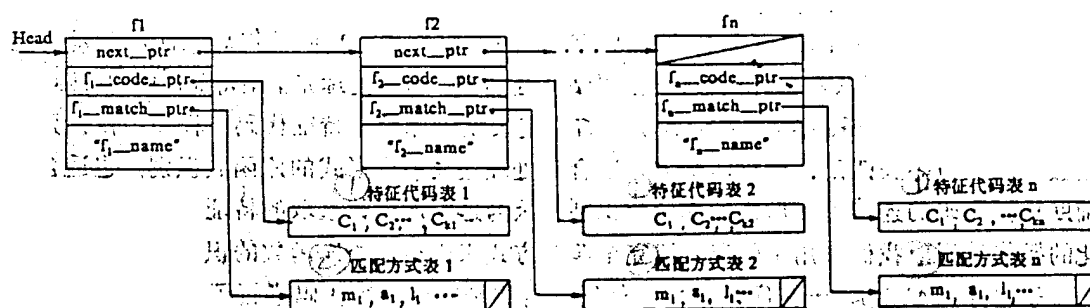sequencing of table elements.

Fig.4 C Library Function Characteristic Code Recognition Table
Structure Schematic

Key: (1) Characteristic Code Table (2) Matching Form Table

13

Library function matching patterns are primarily composed of two sections--library function characteristic code tables and pattern matching form tables. Here, characteristic code tables are composed of recovered characteristic code sequences. Moreover, pattern matching form tables are composed of pattern matching operation coding associated with library function recognition. It includes such information as pattern matching form coding, code matching start addresses in characteristic code tables, code matching length, as well as precise specification forms associated with code matching addresses in object code programs, and so on. For example, matching patterns associated with printf function recognition as shown in Fig.5.

```
   printf          DW          fprintf
                   DW          prt_code
                   DW          prt_match
                   DB          '_printf', 0

   prt_code        DB          055H, 08BH, 0ECH, 083H, 0ECH, 008H

                   DB          057H, 056H, 0BEH, 00EH, 001H, 08DH
                   DB          046H, 006H, 089H, 046H, 0FCH, 056H
                   DB          0E8H, 0BDH, 001H, 083H, 0C4H, 002H
                   DB          ......

   prt_match       DB          01H
                   DW          0, 09H
                   DB          01H
                   DW          0BH, 07H
                   DB          03H
                   DW          12H, 01H, _stbuf
                   DB          01H
                   DW          15H, 0DH
                   DB          03H
                   DW          22H, 01H, _output
                   ...         ...
                   DB          0FFH
```

Fig.5 Matching Patterns Associated with printf Function Recognition

IV.   C LIBRARY FUNCTION PATTERN RECOGNITION REALIZATION

In 8086C decompilation systems, realization methods for C library function pattern recognition techniques are:

1.   Decompilation and Establishment of Library Function Entry Address Tables

In processes associated with program code area decompilation, investigations are made of subprogram transfer commands and entry addresses are found in order to set up library function (including user defined functions) entry address tables utilized in programs, supplying library function recognition uses. First of all, the setting up of library function entry address tables is for the sake of reducing the repetitive nature of library function recognition. The reason is that there are a number of library functions in programs which are capable of being utilized multiple times.

2.   C Library Function Recognition

On the basis of library function entry address tables, library function patterns associated with various entry address start codes and C library function characteristic code recognition tables are taken and sequence matching comparisons are carried out. If matching is successful with patterns of a particular library function, then, the library function name is taken and inserted into the corresponding library function entry address table column. Otherwise, matching continues to be carried out with the next library function pattern in the table. This continues right on until the recognition table is finished. Then, recognition failures enter a "Cannot Recognize" flag into the address table. After library function recognition operations are completed, the work of establishing library function recognition name tables is also finished (see as shown in Fig.6).

At this time, one again takes library function (including user defined function) name character strings in the tables in question and inserts them into the corresponding subprogram transfer commands in compilation language programs, producing compilation language programs carrying library function names.

① 库函数入口地址	② 识别的库函数名
029EH	___chkstk
0037H	__unknown
05E6H	__printf
11A0H	__close
15F2H	__strcmp

Fig.6 Library Function Recognition Name Table (Refer to Fig.1)

Key: (1) Library Function Entry Address   (2) Recognized Library Function Names
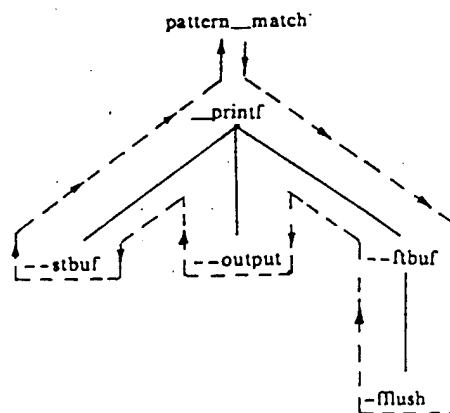

Fig.7 Library Function Pattern Matching Strategy Example

In pattern matching of library functions, with regard to library function patterns having main and supplementary characteristic codes, we opted for the use of depth right prioritized matching strategies, first of all, carrying out main characteristic matching. In conjunction with this, we took supplementary characteristic pattern addresses encountered in matching and compressed them into a queue. After successfully matching main characteristics, then, matching was carried out on supplementary characteristic patterns taken out of feed queues. This continued right along until the queue was empty and stopped.

In matching, as long as there is one instance of matching failure, then, it is a pattern matching failure of the library function in question. Taking printf function as an example, the pattern matching process is as shown in Fig.7. Library function recognition pattern matching algorithms are as shown in Fig.8. In the algorithm in question, only complete matching form, subprogram matching form, and direct section transfer matching form processing methods are given. Other matching forms can be obtained by analogy.

```
Procedure PATTERN_MATCH(SUBR_ADDR. FUNCTION_PATTERN, F_NAME);
begin
  get MATCH_TABLE_ADDR and CODE_TABLE_ADDR from FUNCTION_PATTERN;
  loop
  begin
    get MATCH_FLAG from MATCH_TABLE;
    case MATCH_FLAG
    : end_flag:     begin
                      if stack is empty
                      then begin
                          get F_NAME form FUNCTION_PATTERN;
                          return(true);
                          end
                      esle begin
                          get AUXILIARY_PATTERN from stack;
                          get MATCH_TABLE_ADDR and CODE_TABLE_ADDR
                              from AUXILIARY_PATTERN;
                          get SUBR_ADDR from stack;
                          end
                      end
    ; comp_match:    begin
                      get offset and length from MATCH_TABLE;
                      match_addr = CODE_TABLE_ADDR+offset;
                      prg_addr = SUBR_ADDR+offset;
                      cmp_falg = compare(prg_addr. match_addr. length);
                      if cmp_flag is false
                      then return(false);
                      end;
    ; subr_match:    begin
                      get offset and length from MATCH_TABLE;
                      match_addr = CODE_TABLE_ADDR+offset;
                      prg_addr = SUBR_ADDR+offset;
                      cmp_flag = compare(prg_addr. match_addr. length);
                      if cmp_flag is true
                      then begin
                          get AUXILIARY_PATTERN from MATCH_TABLE;
                          get new_subr_addr by dynamic execution;
                          push new_subr_addr and AUXILIARY_PATTERN into stack;
                          end
                      else return(false);
                      end;
    : jump_match:    begin
                      get offset and length from MATCH_TABLE;
                      match_addr = CODE_TABLE_ADDR+offset;
                      prg_addr = SUBR_ADDR+offset;
                      cmp_flag = compare(prg_addr. match_addr. length);
                      if cmp_flag is true
                      then begin
                          get sub_PATTERN from MATCH_TABLE;
                          get MATCH_TABLE_ADDR and CODE_TABLE_ADDR
                              from sub_PATTERN;
                          get new SUBR_ADDR by dynamic execution;
                          end
                      else return(false);
                      end;
                      ...
    endcase;
  end;
end.
```

Fig.8   Library Function Recognition Pattern Matching Algorithm

18

V.   CONCLUDING REMARKS

In 8086 C decompilation systems, C library function recognition programs already use 8086 compilation language and C language for realization. We already established characteristic code recognition tables having 64 common C library function matching patterns. Experimental program results clearly show that the library functions described above are all capable of being accurately recognized.

As far as the C library function pattern recognition techniques which we studied are concerned, it is possible to expand them to recognize all C library functions which possess independent subprograms. The range of C library functions which can be recognized follows along with enlargements of characteristic code recognition tables and expands. This type of method very greatly reduces the amount of statement translator device code analysis in decompilation systems. This is beneficial for variable data type recovery. It is possible to generalize into decompilation systems associated with other machine types and languages.

Problems which have been discovered now are: how to recognize macro definition functions such as getc and putc, etc. Macro definition functions have corresponding subprograms. However, main body command sequences show up in program code areas causing difficulties in their recognition during decompilation. Our tentative idea is that, after recognizing the subprograms contained in macro definition functions, we opt for the use of methods associated with the recalling of command sequences in recognized program code areas in order to carry out the recognition of macro definition functions.

REFERENCES

〔1〕 刘宗田，朱逸芬，符号执行技术在68000 C反编译程序中的应用，计算机学报，1988，Vol.11，No.10，pp633—637.

〔2〕 蔡元龙编，模式识别，西北电讯工程学院出版社，1986.

〔3〕 Microsoft C 5.0 Reference Book, Microsoft Corporation, 1987.

DISTRIBUTION LIST

DISTRIBUTION DIRECT TO RECIPIENT

ORGANIZATION	MICROFICHE
BO85 DIA/RTS-2FI	1
C509 BALLOC509 BALLISTIC RES LAB	1
C510 R&T LABS/AVEADCOM	1
C513 ARRADCOM	1
C535 AVRADCOM/TSARCOM	1
C539 TRASANA	1
Q592 FSTC	4
Q619 MSIC REDSTONE	1
Q008 NTIC	1
Q043 AFMIC-IS	1
E051 HQ USAF/INET	1
E404 AEDC/DOF	1
E408 AFWL	1
E410 AFDTC/IN	1
E429 SD/IND	1
P005 DOE/ISA/DDI	1
P050 CIA/OCR/ADD/SD	2
1051 AFIT/LDE	1
PO90 NSA/CDB	1
2206 FSL	1

Microfiche Nbr: FTD95C000334
NAIC-ID(RS)T-0023-95